

一份不完整的 clash 教程

楚天杳-ictye

2023 年 3 月 30 日

编版信息

参与制作人列表

名称	贡献	时间
楚天杳	创建和主要编辑者	(空)
(空)	审稿与校订	(空)

目录

1	clash 维基的（不完全）翻译	2
1.1	clash 介绍	2
1.1.1	安装	2
1.2	将 clash 作为服务运行	3
1.3	clash 配置文件	5
1.4	代码示例	20
1.5	外接控制器 api	23
1.6	premium 版本的功能特性	27
1.7	FAQ	34

作者的话

其实我自己也找了很久系统讲解 clash 的文档,但是好像直接介绍 clash 所有配置的人很少,所以最终还是自己做吧。

这份文档主要是翻译 clash 的文档，其实并不复杂，通过机器翻译和我本人的一些修正再加一些修改等等高科技，给大家一个很好的配置 clash 的参考。

文档错误可以直接通过邮箱 northgreen2006@qq.com 进行反馈，我也希望有人能直接改正我的错误。

一些引路的链接

- clash github 项目页: <https://github.com/Dreamacro/clash>
- clash for windows (一个图形界面的 clash, 全平台支持, 傻瓜式操作 (不仅仅是 Windows 系统)。默认是英文的, GitHub 上有很多汉化补丁包, 很好用。): https://github.com/Fndroid/clash_for_windows_pkg
- yacd (一个网页端的 clash 控制面板, 很像 clash for Windows, 但是它是运行浏览器里的, 通过 api 来控制 clash, 很适合在服务器上用): <https://github.com/haishanh/yacdURL>

1 clash 维基的（不完全）翻译

此部分是对 clash 维基文档的一个翻译，稍加作者本人的润色和整理。

1.1 clash 介绍

clash 是一个广受大家喜爱的代理软件，目前分为两个版本：

- clash: 在/Dreamacro/clash上发布的开源软件。
- clash premium: 带有 tun 和其他一些功能的 clash 二进制文件（也是免费的）

1.1.1 安装

直接从<https://github.com/Dreamacro/clash/releases>获取预先构建好的二进制文件，然后将其配置到系统的 patch 中即可（Windows 下最直接的方法是直接复制到 system32），国内用户会有个问题就是Country.mmdb无法下载，这个文件网上也有很多人分享它的链接。当然也可以曲线救国，使

用 clash for Windows 的这个文件，clash for Windows 默认自带这个文件。另外你也可以从源码构建二进制文件，这需要自行查看他们自己的wiki

1.2 将 clash 作为服务运行

序言 clash 被设计成在后台运行，但是目前 golang 下并没有什么完美的方法实现后台守护。我们可以用第三方工具将 clash 设置为守护进程。

systemd 将 clash 的二进制文件拷贝到/usr/local/bin, 将配置文件放在/etc/clash:

```
1 $ cp clash /usr/local/bin
2 $ cp config.yaml /etc/clash/
3 $ cp Country.mmdb /etc/clash/
```

在/etc/systemd/system/clash.service创建配置文件

```
1 [Unit]
2 Description=Clash daemon, A rule-based proxy in Go.
3 After=network.target
4
5 [Service]
6 Type=simple
7 Restart=always
8 ExecStart=/usr/local/bin/clash -d /etc/clash
9
10 [Install]
11 WantedBy=multi-user.target
```

然后你需要重新加载 systemd:

```
1 $ systemctl daemon-reload
```

让 clashd 随系统启动而启动

```
1 $ systemctl enable clash
```

使用以下命令检查 clash 的运行情况和日志

```
1 $ systemctl status clash
2 $ journalctl -xe
```

（本指南的提供者：ktechmidas（#754）

Docker 如果你使用的是 Linux 系统的话我们推荐使用docker compose
在 MacOS 或者 Windows 上更推荐使用第三方 clash GUI（ClashX
Pro或者clash for Windows）

另外并不推荐在 Dockers 容器中运行 Clash Premium（#2249）

```
1 services :
2   clash:
3     # ghcr.io/dreamacro/clash
4     # ghcr.io/dreamacro/clash-premium
5     # dreamacro/clash
6     # dreamacro/clash-premium
7     image: dreamacro/clash
8     container_name: clash
9     volumes:
10      - ./config.yaml:/root/.config/clash/config.yaml
11      # - ./ui:/ui # dashboard volume
12     ports:
13      - "7890:7890"
14      - "7891:7891"
15      # - "8080:8080" # 外部控制器
16     # TUN
17     # cap_add:
18     #   - NET_ADMIN
19     # devices:
20     #   - /dev/net/tun
21     restart: unless-stopped
22     network_mode: "bridge" # 或者是linux上的host
```

保存为docker-compose.yaml然后在同一目录下创建config.yaml，并且运行以下命令启动 clash

```
1 $ docker-compose up -d
```

使用以下命令查看日志

```
1 $ docker-compose logs
```

使用以下命令停止 clash

```
1 $ docker-compose stop
```

PM2 PM2

```
1 $ wget -qO- https://getpm2.com/install.sh | bash
```

```
2 $ pm2 start clash
```

1.3 clash 配置文件

简介 clash 使用YAML语言, (YAML Ain't Markup Language) 写配置文件, YAML 语言旨在上易于计算机的解析和人的阅读。在这个章节里我们将会介绍 clash 的常见功能并且如何使用和配置它们。

clash 通过在本地创建一个 HTTP、SOCKS5 或者透明代理服务器, 当一个请求 (或者说是数据包) 被收到时, clash 将会将数据路由到不同的远程代理服务器, 这些节点使用 VMess、Shadowsocks、Snell、Trojan、SOCKS5 或 HTTP 协议。

所有的配置选项 这里将会列出所有的配置选项。你只需要照着做就行了。

```
1 # 本地HTTP (S) 代理服务器的端口
2 port: 7890
3
4 # 本地SOCKS5代理服务器的端口
5 socks-port: 7891
6
7 # 在Linux和macOS上的透明代理服务器的端口 (重定向TCP和TProxy
  UDP)
```

```
8 # redir-port: 7892
9
10 # Linux下的透明服务器端口（TProxy TCP和TProxy UDP）
11 # tproxy-port: 7893
12
13 # 在同一端口上使用HTTP（S）和SOCKS4（A）
14 # mixed-port: 7890
15
16 # 本地SOCKS5/HTTP（S）代理服务器的验证密钥
17 # authentication:
18 # - "user1:pass1"
19 # - "user2:pass2"
20
21 # 设置为true，以允许来自其他局域网IP地址的连接到本地端服务器。
22 # allow-lan: false
23
24 # 只在“allow-lan”为“true”的情况下可用
25 # 这个选项控制那些局域网ip可以链接到本机。
26 # '*': 允许所有IP地址
27 # 192.168.122.11: 允许一个IPv4地址
28 # "[aaaa::a8aa:ff:fe09:57d8]": 允许一个IPv6地址
29 # bind-address: '*'
30
31 # clash路由策略
32 # rule: 基于规则的数据包路由模式
33 # global: 所有的数据包都将转发到单个规则
34 # direct: 直接向互联网转发数据包
35 mode: rule
36
37 # 一般情况下，clash会将日志输出到标准输入输出流（STDOUT）
38 # 可选参数: info / warning / error / debug / silent
39 # log-level: info
40
```

```
41 # 当这个选项被设置为false时，解释器将不会使用NAT-IPv6
42 # ipv6: false
43
44 # RESTful网络API监听地址。通过这个功能，你能够控制或者开发一个
    clash的web控制端
45 external-controller: 127.0.0.1:9090
46
47 # 在配置目录的相对路径或放置一些静态web资源的目录的绝对路径。
48 # Clash的核心将会在“http://{ { external controller } }/ui”
49 # 上为其提供web服务器。
50 # 通过这个，你可以将yacd等可以与clash RESTful API对接的网页程序部
    署
51 # 于此，这是很方便的。
52 # external-ui: 你的路径
53
54 # RESTful API的密钥（可选）
55 # 要通过HTTP头‘Authorization: Bearer $ {secret}’进行身份验证
56 # 如果RESTful api 在0.0.0.0监听，则要求验证
57 # secret: ""
58
59 # 接口名字（Outbound interface name）
60 # interface-name: en0
61
62 # fmark只有在Linux下是可用的
63 # routing-mark: 6666
64
65 # 静态DNS解析（像/etc/host）
66 #
67 # 支持通配符（e.g. *.clash.dev, *.foo.*.example.com）
68 # 非通配符域名优先级高于通配符域名
69 # e.g. foo.example.com > *.example.com > .example.com
70 # P.S. +.foo.com 和 .foo.com 和 foo.com 是平等的。
71 # hosts:
```

```
72 # '*.clash.dev': 127.0.0.1
73 # '.dev': 127.0.0.1
74 # 'alpha.clash.dev': '::1'
75
76 # profile:
77 # 将'select'的结果保存到$HOME/.config/clash/.cache
78 # 如果你并不需要这个，请将它设为false
79 # 当两个不同的配置具有相同名称的组时，所选择的值将被共享
80 # store-selected: true
81
82 # 保持fakeip
83 # store-fake-ip: false
84
85 # DNS服务器设置
86 # 这个选项是可选的，如果没有，将不会设置DNS服务器。
87 dns:
88   enable: false
89   listen: 0.0.0.0:53
90   # ipv6: false # 当为false时，对AAAA的请求的响应将为空
91
92   # DNS服务器使用以下名称服务器进行解析
93   # 只需要指定IP地址
94   default-nameserver:
95     - 114.114.114.114
96     - 8.8.8.8
97   # enhanced-mode: fake-ip
98   fake-ip-range: 198.18.0.1/16 # CIDR假IP地址池
99   # use-hosts: true # 查找地址并且返回IP记录
100
101   # 访问这里的IP将不会使用虚假IP
102   # 也就是说将会向这些IP的访问都会使用
103   # 真实IP地址
104   # fake-ip-filter:
```



```
105 # - '*.lan'
106 # - localhost.ptlogin2.qq.com
107
108 # 支持UDP, TCP, DoT, DoH协议。 你能够指定端口。
109 # 所有的DNS请求都不经代理发送到解析服务器, Clash将会采取第一个
    响应, 并不会综合起来。
110 nameserver:
111     - 114.114.114.114 # default value
112     - 8.8.8.8 # default value
113     - tls://dns.rubyfish.cn:853 # TLS的DNS
114     - https://1.1.1.1/dns-query # HTTPS的DNS
115     - dhcp://en0 # 来自DHCP的DNS
116     # - '8.8.8.8#en0'
117
118 # 有'fallback'选项时, DNS服务器将向此段中的服务器以及'
    nameservers'中的服务器发送并发请求
119 # 当GEOIP国家地理位置不是'CN'(中国)时, 将使用'fallback'服务器
    中的回应。
120 # fallback:
121 # - tcp://1.1.1.1
122 # - 'tcp://1.1.1.1#en0'
123
124 # 如果使用'nameservers'服务器解析得到的IP地址属于指定的子网范围
    内, 则被视为无效地址, DNS服务器将使用'fallback'服务器返回解析
    结果。
125 # IP address resolved with servers in 'nameserver' is used when '
    fallback-filter.geoip' is true and when GEOIP of the IP address
    is 'CN': 当'fallback-filter.geoip'为true且IP地址的GEOIP为'CN'
    时, 将使用在'nameserver'中解析的服务器的IP地址。
126 #
127
128 #如果'fallback-filter.geoip'为false, 且不匹配'fallback-filter.ipcidr'
    , 则始终使用来自'nameserver'命名服务器的结果。
```

```
129 # 这是对DNS污染攻击的一种防范措施。
130 # fallback-filter :
131 #   geoip: true
132 #   geoip-code: CN
133 #   ipcidr:
134 #     - 240.0.0.0/4
135 #   domain:
136 #     - '+.google.com'
137 #     - '+.facebook.com'
138 #     - '+.youtube.com'
139
140 # 通过特定的DNS服务器解析指定域名。
141 # nameserver-policy:
142 #   'www.baidu.com': '114.114.114.114'
143 #   '+.internal.crop.com': '10.0.0.1'
144
145 proxies:
146 # Shadowsocks
147 # 密码支持(加密方法):
148 #   aes-128-gcm aes-192-gcm aes-256-gcm
149 #   aes-128-cfb aes-192-cfb aes-256-cfb
150 #   aes-128-ctr aes-192-ctr aes-256-ctr
151 #   rc4-md5 chacha20-ietf xchacha20
152 #   chacha20-ietf-poly1305 xchacha20-ietf-poly1305
153 - name: "ss1"
154   type: ss
155   server: server
156   port: 443
157   cipher: chacha20-ietf-poly1305
158   password: "password"
159   # udp: true
160
161 - name: "ss2"
```

```

162     type: ss
163     server: server
164     port: 443
165     cipher: chacha20-ietf-poly1305
166     password: "password"
167     plugin: obfs
168     plugin-opts:
169         mode: tls # or http
170         # host: bing.com
171
172 - name: "ss3"
173     type: ss
174     server: server
175     port: 443
176     cipher: chacha20-ietf-poly1305
177     password: "password"
178     plugin: v2ray-plugin
179     plugin-opts:
180         mode: websocket # 暂时没有QUIC
181         # tls: true # wss
182         # skip-cert-verify: true
183         # host: bing.com
184         # path: "/"
185         # mux: true
186         # headers:
187         #     custom: value
188
189     # vmess
190     # 密码支持 auto/aes-128-gcm/chacha20-poly1305/none
191 - name: "vmess"
192     type: vmess
193     server: server
194     port: 443

```

```
195     uuid: uuid
196     alterId: 32
197     cipher: auto
198     # udp: true
199     # tls: true
200     # skip-cert-verify: true
201     # servername: example.com # 优先于wss主机
202     # network: ws
203     # ws-opts:
204     #   path: /path
205     #   headers:
206     #     Host: v2ray.com
207     #   max-early-data: 2048
208     #   early-data-header-name: Sec-WebSocket-Protocol
209
210   - name: "vmess-h2"
211     type: vmess
212     server: server
213     port: 443
214     uuid: uuid
215     alterId: 32
216     cipher: auto
217     network: h2
218     tls: true
219     h2-opts:
220       host:
221         - http.example.com
222         - http-alt.example.com
223       path: /
224
225   - name: "vmess-http"
226     type: vmess
227     server: server
```

```
228     port: 443
229     uuid: uuid
230     alterId: 32
231     cipher: auto
232     # udp: true
233     # network: http
234     # http-opts:
235     #   # method: "GET"
236     #   # path:
237     #   #   - '/'
238     #   #   - '/video'
239     #   # headers:
240     #   #   Connection:
241     #   #     - keep-alive
242
243   - name: vmess-grpc
244     server: server
245     port: 443
246     type: vmess
247     uuid: uuid
248     alterId: 32
249     cipher: auto
250     network: grpc
251     tls: true
252     servername: example.com
253     # skip-cert-verify: true
254     grpc-opts:
255       grpc-service-name: "example"
256
257   # socks5
258   - name: "socks"
259     type: socks5
260     server: server
```

```
261     port: 443
262     # username: username
263     # password: password
264     # tls: true
265     # skip-cert-verify: true
266     # udp: true
267
268     # http
269     - name: "http"
270     type: http
271     server: server
272     port: 443
273     # username: username
274     # password: password
275     # tls: true # https
276     # skip-cert-verify: true
277     # sni: custom.com
278
279     # Snell
280     # 注意,暂时还不支持UDP协议
281     - name: "snell"
282     type: snell
283     server: server
284     port: 44046
285     psk: yourpsk
286     # version: 2
287     # obfs-opts:
288     #   mode: http # 或者tls
289     #   host: bing.com
290
291     # Trojan
292     - name: "trojan"
293     type: trojan
```

```
294     server: server
295     port: 443
296     password: yourpsk
297     # udp: true
298     # sni: example.com # 也可以成为服务器名称
299     # alpn:
300     #   - h2
301     #   - http/1.1
302     # skip-cert-verify: true
303
304   - name: trojan-grpc
305     server: server
306     port: 443
307     type: trojan
308     password: "example"
309     network: grpc
310     sni: example.com
311     # skip-cert-verify: true
312     udp: true
313     grpc-opts:
314       grpc-service-name: "example"
315
316   - name: trojan-ws
317     server: server
318     port: 443
319     type: trojan
320     password: "example"
321     network: ws
322     sni: example.com
323     # skip-cert-verify: true
324     udp: true
325     # ws-opts:
326     #   path: /path
```

```

327     # headers:
328     # Host: example.com
329
330     # ShadowsocksR
331     # 支持的密码（加密方式）:ss中所有的流密码
332     # 支持的混淆方式（obfses）:
333     # plain http_simple http_post
334     # random_head tls1.2_ticket_auth tls1.2_ticket_fastauth
335     # The supported supported protocols:
336     # origin auth_sha1_v4 auth_aes128_md5
337     # auth_aes128_sha1 auth_chain_a auth_chain_b
338     - name: "ssr"
339     type: ssr
340     server: server
341     port: 443
342     cipher: chacha20-ietf
343     password: "password"
344     obfs: tls1.2_ticket_auth
345     protocol: auth_sha1_v4
346     # obfs-param: domain.tld
347     # protocol-param: "#"
348     # udp: true
349
350 proxy-groups:
351     # relay chains the proxies. proxies shall not contain a relay. No
      UDP support.
352     # Traffic: clash <-> http <-> vmess <-> ss1 <-> ss2 <->
      Internet
353     - name: "relay"
354     type: relay
355     proxies:
356         - http
357         - vmess

```



```
358     - ss1
359     - ss2
360
361     # url-test会通过对代理进行URL的标准测试进行测速来选择使用哪个代理
362     - name: "auto"
363     type: url-test
364     proxies:
365         - ss1
366         - ss2
367         - vmess1
368     # tolerance: 150
369     # lazy: true
370     url: 'http://www.gstatic.com/generate_204'
371     interval: 300
372
373     # fallback会按照优先级选择一个可用的策略。通过一个URL对代理服务
        器进行测试，就像url-test组一样。
374     - name: "fallback-auto"
375     type: fallback
376     proxies:
377         - ss1
378         - ss2
379         - vmess1
380     url: 'http://www.gstatic.com/generate_204'
381     interval: 300
382
383     # 负载均衡：相同的eTLD+1请求将被拨打到同一个代理。 load-
        balance: The request of the same eTLD+1 will be dial to the same
        proxy.
384     - name: "load-balance"
385     type: load-balance
386     proxies:
```

```
387     - ss1
388     - ss2
389     - vmess1
390     url: 'http://www.gstatic.com/generate_204'
391     interval: 300
392     # strategy: consistent-hashing # 或者 round-robin
393
394     # select用于选择代理或代理组。建议使用RESTful API切换代理，适合
      在GUI中的使用。
395     - name: Proxy
396       type: select
397       # disable-udp: true
398       proxies:
399         - ss1
400         - ss2
401         - vmess1
402         - auto
403
404     # 可以将流量定向到另一个接口名或者fwmark，也支持在代理中使用。
405     - name: en1
406       type: select
407       interface -name: en1
408       routing-mark: 6667
409       proxies:
410         - DIRECT
411
412     - name: UseProvider
413       type: select
414       use:
415         - provider1
416       proxies:
417         - Proxy
418         - DIRECT
```

```
419
420 proxy-providers:
421   provider1:
422     type: http
423     url: "url"
424     interval: 3600
425     path: ./provider1.yaml
426     health-check:
427       enable: true
428       interval: 600
429       # lazy: true
430       url: http://www.gstatic.com/generate_204
431   test:
432     type: file
433     path: /test.yaml
434     health-check:
435       enable: true
436       interval: 36000
437       url: http://www.gstatic.com/generate_204
438
439 tunnels:
440   # 一行的配置
441   - tcp/udp,127.0.0.1:6553,114.114.114.114:53,proxy
442   - tcp,127.0.0.1:6666,rds.mysql.com:3306,vpn
443   # 完整的yaml配置
444   - network: [tcp, udp]
445     address: 127.0.0.1:7777
446     target: target.com
447     proxy: proxy
448
449 rules:
450   - DOMAIN-SUFFIX,google.com,auto
451   - DOMAIN-KEYWORD,google,auto
```

```

452 - DOMAIN,google.com,auto
453 - DOMAIN-SUFFIX,ad.com,REJECT
454 - SRC-IP-CIDR,192.168.1.201/32,DIRECT
455 # 可选的参数 “no-resolve” 是针对IP规则的
456 - IP-CIDR,127.0.0.0/8,DIRECT
457 - GEOIP,CN,DIRECT
458 - DST-PORT,80,DIRECT
459 - SRC-PORT,7777,DIRECT
460 - RULE-SET,apple,REJECT # 仅限于Premium版本
461 - MATCH,auto

```

1.4 代码示例

这一章节在于给大家一些特殊的示例用法，给大家展示一下这个软件的强大的特殊用法。

基于规则的 wireguard 此功能只能在支持 wireguard 并且开启此功能的内核上使用。Table选项能够阻止wg-quick覆盖默认路由

example "wg0.conf"

```

1 [ Interface ]
2 PrivateKey = ...
3 Address = 172.16.0.1/32
4 MTU = ...
5 Table = 6666
6 PostUp = ip rule add from 172.16.0.1/32 table 6666
7
8 [Peer]
9 AllowedIPs = 0.0.0.0/0
10 AllowedIPs = ::/0
11 PublicKey = ...
12 Endpoint = ...

```

那么在 Clash 中，你只需要创建一个名为“DIRECT”的具有特输出接口代理组。

```
1 proxy-groups:
2   - name: Wireguard
3     type: select
4     interface -name: wg0
5     proxies:
6       - DIRECT
7 rules:
8   - DOMAIN,google.com,Wireguard
```

与 OpenConnect 一起使用 OpenConnect 支持 Cisco AnyConnect SSL VPN, Juniper Network Connect, Palo Alto Networks (PAN) GlobalProtect SSL VPN, Pulse Connect Secure SSL VPN, F5 BIG-IP SSL VPN, FortiGate SSL VPN and Array Networks SSL VPN.

例如，当你的公司里使用 Cisco AnyConnect 进行内部网络访问。在这里我会展示如何使用 clash 的路由策略进行更方便的上网。

首先，你需要安装vpn-slice，这个工具能够覆盖默认的 OpenConnect 行为。通俗的来说，这个软件能够阻止 VPN 软件接管你的默认网络路由。

接下来你需要写一个脚本（在下文中我们将其称作为tun0.sh）就像这样子：(windows 下请自行脑补)

```
1 #!/bin/bash
2 ANYCONNECT_HOST="vpn.example.com"
3 ANYCONNECT_USER="john"
4 ANYCONNECT_PASSWORD="foobar"
5 ROUTING_TABLE_ID="6667"
6 TUN_INTERFACE="tun0"
7
8 #如果服务器在中国大陆，则需要添加--no-dtls。因为中国的UDP可能
   是不稳定的。
9 echo "$ANYCONNECT_PASSWORD" | \
10  openconnect \
```

```

11  --non-inter \
12  --passwd-on-stdin \
13  --protocol=anyconnect \
14  --interface $TUN_INTERFACE \
15  --script "vpn-slice
16  if [ \"$reason\" = 'connect' ]; then
17      ip rule add from \"$INTERNAL_IP4_ADDRESS\" table
          $ROUTING_TABLE_ID
18      ip route add default dev \"$TUNDEV\" scope link table
          $ROUTING_TABLE_ID
19  elif [ \"$reason\" = 'disconnect' ]; then
20      ip rule del from \"$INTERNAL_IP4_ADDRESS\" table
          $ROUTING_TABLE_ID
21      ip route del default dev \"$TUNDEV\" scope link table
          $ROUTING_TABLE_ID
22  fi \" \
23      --user $ANYCONNECT_USER \
24      https://$ANYCONNECT_HOST

```

然后你需要将其配置为一个 systemd 服务。

创建 `text/etc/systemd/system/tun0.service`

```

1  Description=Cisco AnyConnect VPN
2  After=network-online.target
3  Conflicts=shutdown.target sleep.target
4
5  [Service]
6  Type=simple
7  ExecStart=/path/to/tun0.sh
8  KillSignal=SIGINT
9  Restart=always
10 RestartSec=3
11 StartLimitIntervalSec=0
12

```

```

13 [ Install ]
14 WantedBy=multi-user.target

```

然后启动这个服务

```

1 chmod +x /path/to/tun0.sh
2 systemctl daemon-reload
3 systemctl enable tun0
4 systemctl start tun0

```

然后你可以查看日志查看它有没有正常运行,简单的看是否创建了tun0端口。

和 Wireguard 相似,, 将出口连接到 TUN 设备只需要添加一个代理组:

```

1 proxy-groups:
2   - name: Cisco AnyConnect VPN
3     type: select
4     interface --name: tun0
5     proxies:
6       - DIRECT

```

.....然后就可以准备使用了。添加需要的规则:

```

1 rules:
2   - DOMAIN-SUFFIX,internal.company.com,Cisco AnyConnect VPN

```

当出现一些问题时, 你可以查看调试级别的日志。

1.5 外接控制器 api

介绍 外部控制器能够让用户程序化地控制 clash 通过 HTTP RESTful API。很大程度上,第三方的 clash 图形界面都是基于这个功能。通过external-controller指定一个地址开启这个功能。

TIP: 此文档不算太完善,请看<http://clash.gitbook.io/doc/restful-api>。

身份验证

- 外部控制器接受 Bearer Tokens 作为验证访问的方式。
 - 使用 Bearer <你的密钥> 作为请求的标头，以便传递网络凭据。

RESTful API 文档

日志 (Logs)

- /logs
 - 请求方法: GET
 - * 完整路径: GET /logs
 - * 说明: 得到实时的日志

流量 (Traffic)

- /traffic
 - 请求方法: GET
 - * 完整路径: GET /traffic
 - * 描述: 获取实时的流量数据

版本 (Version)

- /version
 - 请求方法: GET
 - * 完整路径: GET /version
 - * 描述: 获得 clash 版本

配置 (Configs)

- /configs
 - 请求方法: GET
 - * 完整路径: GET /configs
 - * 描述: 获取基础的配置信息

- 请求方法: PUT
 - * 完整路径: PUT /configs
 - * 描述: 重新加载基础配置
- 请求方法: PATCH
 - * 完整路径: PATCH /configs
 - * 描述: 更新基础配置

代理 (proxies)

- /proxies
 - 请求方法: GET
 - * 完整路径: GET /proxies
 - * 描述: 获得特定的代理信息
 - 请求方法: PUT
 - * 完整路径: PUT /proxies/:name
 - * 描述: 选择具体的代理
- /proxies/:name/delay
 - 请求方法: GET
 - * 完整路径: GET /proxies/:name/delay
 - * 描述: 获取特定的代理的延迟测试信息

规则 (Rules)

- /rules
 - 请求方法: GET
 - * 完整路径: GET /rules
 - * 得到规则的详细信息

连接 (connections)

- /connections
 - 请求方法: GET
 - * 完整路径: GET /connections
 - * 描述: 获取连接信息
 - 请求方法: DELETE
 - * 完整路径: DELETE /connections
 - * 描述: 关闭所有链接
- /connections/:id
 - 请求方法: DELETE
 - * 完整路径: DELETE /connections/:id
 - * 描述: 关闭特定连接

供应商 (Providers)

- /providers/proxies
 - 请求方法: GET
 - * 完整路径: GET /providers/proxies
 - * 描述: 获取所有的代理的信息
- /providers/proxies/:name
 - 请求方法: GET
 - * 完整路径: GET /providers/proxies/:name
 - * 描述: 获取特定的代理提供商的代理的信息
 - PUT
 - * 完整路径: PUT /providers/proxies/:name
 - * 描述: 选择特定的代理提供商
- /providers/proxies/:name/healthcheck
 - 请求方法: GET
 - * 完整路径: GET /providers/proxies/:name/healthcheck
 - * 描述: 获取特定的代理提供商的代理信息

DNS 查询 (DNS Query)

- /dns/query
 - 请求方法: GET
 - * 完整路径GET /providers/proxies
 - * 描述: 获取指定名称和类型的 DNS 查询数据
 - * 参数:
 - name (必选): 所要查询的域名
 - type (可选): 查询 DNS 的记录类型 (例如: A, MX, CNAME 等) 如果没有指定的话默认是A
 - 示例: GET /dns/query?name=example.com&type=A

1.6 premium 版本的功能特性

premium 核心是专有的，在内部 ci 管线上构建

代理 由于对 gvisor 的依赖，目前只有 premium 版本有 Wireguard

```

1 proxies:
2   - name: "wg"
3     type: wireguard
4     server: 127.0.0.1
5     port: 443
6     ip: 172.16.0.2
7     # ipv6: your_ipv6
8     private-key: eCtXsJZ27+4
9       PbhDkHnB923tkUn2Gj59wZw5wFA75MnU=
10    public-key: Cr8hWlKvtDt7nrvf+f0brNQQzabAqrjfBvas9pmowjo
11      =
12    # preshared-key: base64
13    # remote-dns-resolve: true # 使用 'dns' 字段对 DNS 进行远程解
14      析,
15    默认是 true

```

```
13     # dns: [1.1.1.1, 8.8.8.8]
14     # mtu: 1420
15     udp: true
```

TUN 驱动 不同于硬件物理网卡，TUN 是完全由软件实现的虚拟网络设备，在功能上 TUN 和物理网卡没有区别，它们同样都是网络设备，都可以设置 IP 地址，而且都属于网络设备管理模块，由网络设备管理模块统一来管理。（说人话就是虚拟网卡）

开启 TUN 后，clash 就能以虚拟网卡的方式就能接管你设备的所有网络流量。

需要开启 TUN 的话只需要将以下内容添加到配置文件中

```
1 tun:
2   enable: true
3   stack: system # 或者 gvisor
4   # dns-hijack:
5   #   - 8.8.8.8:53
6   #   - tcp://8.8.8.8:53
7   #   - any:53
8   #   - tcp://any:53
9   auto-route: true # 自动设置全局路由
10  auto-detect-interface: true # 与 interface-name 冲突
```

或者

```
1 interface-name: en0
2
3 tun:
4   enable: true
5   stack: system # or gvisor
6   # dns-hijack:
7   #   - 8.8.8.8:53
8   #   - tcp://8.8.8.8:53
9   auto-route: true # auto set global route
```

对于 DNS 服务器推荐使用fake-ip模式

clash 需要更高的权限创建 TUN 设备

```
$sudo ./clash
```

然后手动创建默认路由和 DNS 服务器。如果你的设备已经有 TUN 设备，那么 Clash TUN 可能无法正常工作。在这种情况下，可以使用 fake-ip-filter 解决问题。

TIPS: tun 驱动也能在安卓设备上使用,但是它的控制设备是 /dev/tun 而非 /dev/net/tun, 所以你需要创建一个软连接解决这个问题, 例如:

```
ln -sf /dev/tun /dev/net/tun
```

NOTE: auto-route和auto-detect-interface 仅在 macOS、Windows, Linux 和 Android 下可用, 接收 IPv4 流量

windows 下的 TUN 设置 到<https://www.wintun.net>下载最后的发行版, 拷贝正确的wintun.dll到 clash 的目录下

```
1 tun:
2   enable: true
3   stack: gvisor # 或者 system
4   dns-hijack:
5     - 198.18.0.2:53 # 当 'fake-ip-range' 是 198.18.0.1/16, 应劫持
6     198.18.0.2:53
7   auto-route: true # 为Windows自动设置全局路由
8   # 建议使用 'interface-name'
9   auto-detect-interface: true # 自动检测接口, 与 'interface-name'
10 冲突
```

最后打开 clash 即可。

脚本 脚本能够让用户以编程的方式灵活的选择网络流量包的代理方式。

例如:

```
1 mode: Script
2
3 # 参考 https://lancellc.gitbook.io/clash/clash-config-file/script
4 script :
5   code: |
6     def main(ctx, metadata):
7       ip = metadata["dst_ip"] = ctx.resolve_ip(metadata["host"])
8       if ip == "":
9         return "DIRECT"
10
11       code = ctx.geoip(ip)
12       if code == "LAN" or code == "CN":
13         return "DIRECT"
14
15       return "Proxy" # 对于没有被其他脚本匹配的请求的默认策略。
```

NOTE: 如果你想使用 IP 规则 (IP-CIDR GEOIP), 你需要先手动解析 IP 并且将它分配到元数据里。

上下文和元数据:

```
1 interface Metadata {
2   type: string // socks5、http
3   network: string // tcp
4   host: string
5   src_ip: string
6   src_port: string
7   dst_ip: string
8   dst_port: string
9 }
10
11 interface Context {
12   resolve_ip: (host: string) => string // ip string
```

```

13 resolve__process__name: (metadata: Metadata) => string
14 resolve__process__path: (metadata: Metadata) => string
15 geoip: (ip: string) => string // country code
16 log: (log: string) => void
17 proxy__providers: Record<string, Array<{ name: string, alive: boolean,
18 delay: number }>>
19 rule__providers: Record<string, { match: (metadata: Metadata) =>
    boole
20 an }>
21 }

```

脚本快捷方式 在rules里使用脚本

NOTE: src_port和dst_port都被定义为变量

```

1 script :
2   shortcuts:
3     quic: network == 'udp' and dst_port == 443
4
5 rules :
6   - SCRIPT,quic,REJECT

```

功能 这部分介绍它的功能

```

1 type resolve_ip = (host: string) => string // ip 字符串
2 type in_cidr = (ip: string, cidr: string) => boolean // cidra中的ip
3 type geoip = (ip: string) => string // 国家代号
4 type match_provider = (name: string) => boolean // 在规则配置文件里
5 type resolve_process_name = () => string // 查找进程名 (curl .e.g)
6 type resolve_process_path = () => string // 查找进程路径 (/usr/bin/
    curl
7 .e.g)

```

规则配置文件 对于代理程序来说，规则配置文件都是等价的，它允许用户加载其他文件的规则并且使整体配置文件更加简洁。这也是一个仅限于 Premium 的功能

定义一个规则配置文件需要添加`rule_providers`字段到主配置文件：

```
1 rule-providers:
2   apple:
3     behavior: "domain"
4     type: http
5     url: "url"
6     interval: 3600
7     path: ./apple.yaml
8   microsoft:
9     behavior: "domain"
10    type: file
11    path: /microsoft.yaml
```

有三种可用的`behavior`：

domain 域名

```
1 payload:
2   - '.blogger.com'
3   - '*.*.microsoft.com'
4   - 'books.itunes.apple.com'
```

ipcidr ip 地址

```
1 payload:
2   - '192.168.1.0/24'
3   - '10.0.0.0.1/32'
```


classical 传统的

```
1 payload:
2   - DOMAIN-SUFFIX,google.com
3   - DOMAIN-KEYWORD,google
4   - DOMAIN,ad.com
5   - SRC-IP-CIDR,192.168.1.201/32
6   - IP-CIDR,127.0.0.0/8
7   - GEOIP,CN
8   - DST-PORT,80
9   - SRC-PORT,7777
10  # 这里不需要MATCH

1 # 仅限于Premium
2 rule-providers:
3   apple:
4     behavior: "domain" # domain, ipcidr 或者 classical (仅限于premium
        核心 )
5     type: http
6     url: "url"
7     interval: 3600
8     path: ./apple.yaml
9   microsoft:
10    behavior: "domain"
11    type: file
12    path: /microsoft.yaml
13
14 rules:
15   - RULE-SET,apple,REJECT
16   - RULE-SET,microsoft,policy
```

追踪 <https://github.com/Dreamacro/clash-tracing>

```

1 profile :
2   tracing: true

```

eBPF 这个需要内核支持,只能捕获 NIC 的输出流量并且和**auto-route**有冲突。

```

1 ebpf:
2   redirect --to-tun:
3     -- eth0

```

自动重定向 单纯在 Go 语言上使用 Linux 内核的 nftables 的特性。它可以在不进行任何网络配置的情况下使用**redir-port**(TCP)。

建议使用 TUN 来处理 TCP 流量。与仅使用 TUN 相比,它提高了一些低性能设备的网络吞吐量性能。

```

1 interface --name: en0
2
3 tun:
4   enable: true
5   stack: system
6   dns-hijack:
7     -- any:53
8   auto-redir: true
9   auto-route: true

```

1.7 FAQ

error:unsupported rule type RULE-SET (不支持的规则类型 RULE-SET)

```

1 FATA[0000] Parse config error : Rules [0] [RULE-SET,apple,REJECT] error
  : unsupported rule type RULE-SET

```

你用的是 clash 的开源版本，但是 Rule Providers（规则提供程序）目前只能在 Premium（免费的）内核版本上使用

我想要 VLESS 支持 概述：不可能，绝对不可能（bushi）论述：<https://github.com/Dreamacro/clash/issues/1185>